Implementing Sun® Microsystem's Java™ Pet Store J2EE™ Blueprint Application using Microsoft® .NET

Version 1.5

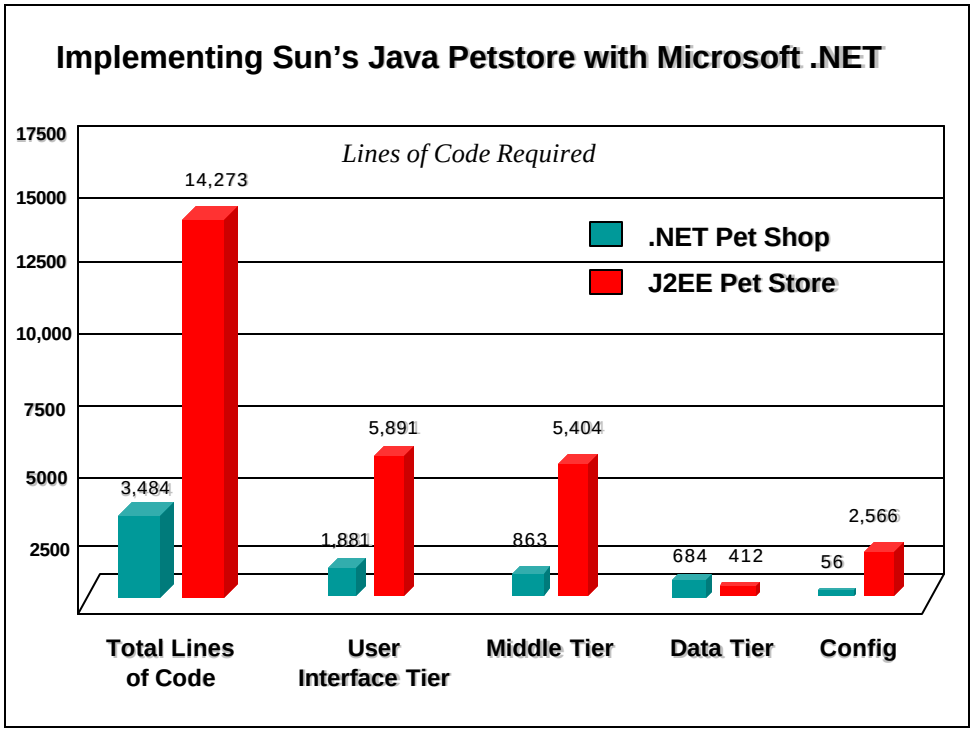
November, 2001

Contents

Abstract

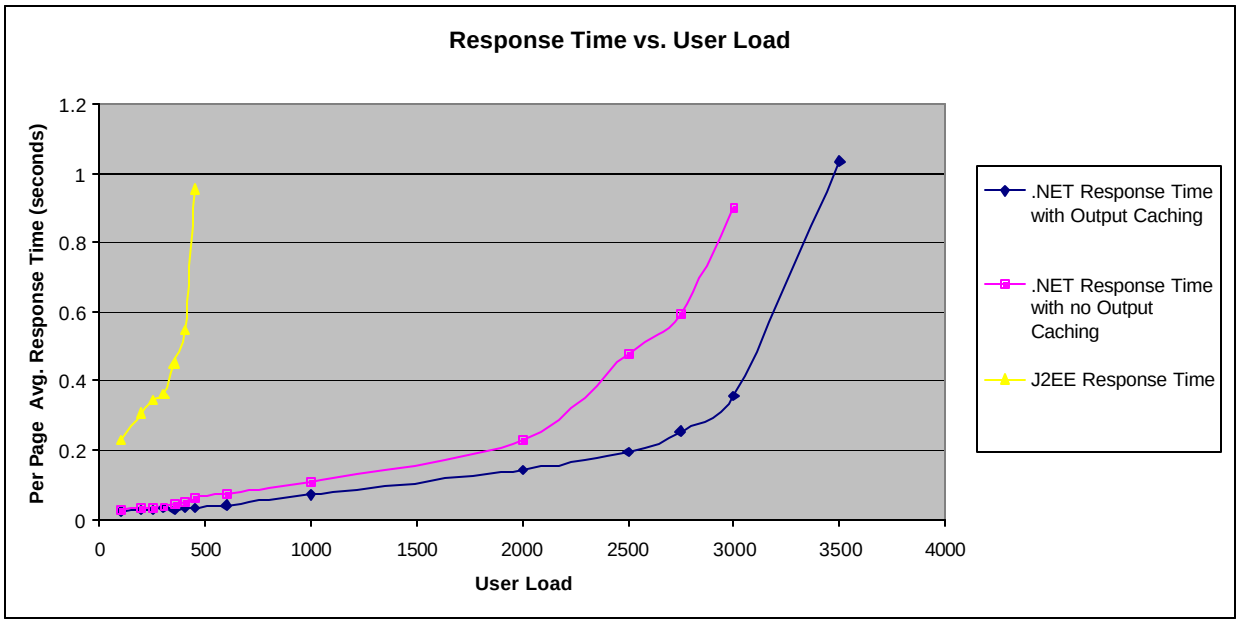
The purpose of this study is to take Sun's primary J2EE blueprint application, the Sun Java Pet Store, and implement the same functionality using Microsoft .NET. Based on the .NET version of Sun's J2EE best-practice sample application, customers can directly compare Microsoft .NET to J2EE-based application servers across a variety of fronts. In addition, the study compares the architecture and programming models of each platform to evaluate relative developer productivity. This includes a side-by-side comparison of the objects, files and lines of code required to implement the same functionality in each platform across all three tiers of the application. The study also shows how the .NET Pet Shop application can be easily extended to include support for XML Web Services (based on the SOAP and UDDI standards) as well as support for mobile devices. Full source code to the .NET implementation is available at *http://www.gotdotnet.com/compare*. Source code to the Java Pet Store application is publicly available at the Java 2 Platform Enterprise Edition Blueprints website: *http://java.sun.com/j2ee/blueprints/index.html*.

Comparison of Lines of Code Required[1]

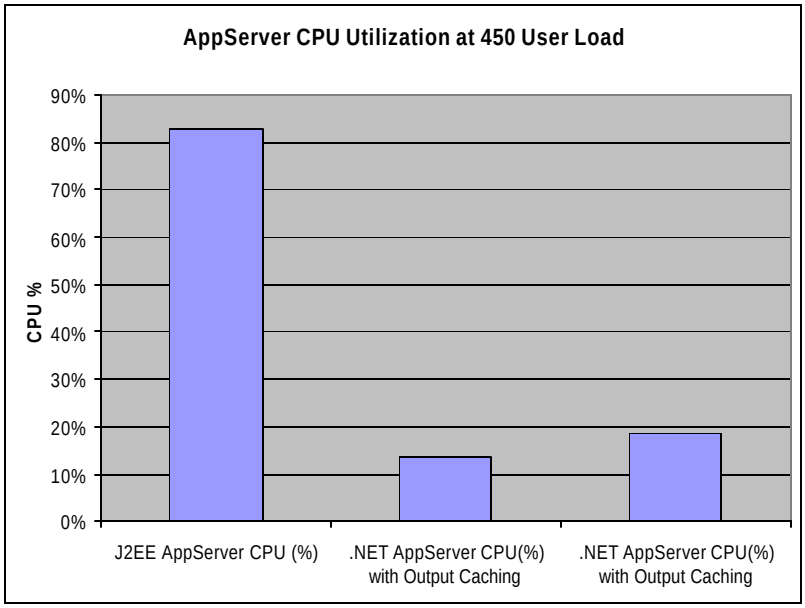


[1] This is a code count comparing the implementation logic of the exact same application functionality for each platform. Full details on the precise code count are provided later in this document and in Appendix 1.

Comparison of Performance and Scalability[2]



Comparison of CPU % Resources Required[3]



[2,3] Based on Oracle's published benchmark data for a fully tuned and optimized version of Java Pet Store published as part of their Oracle 9i Application Server Challenge. See *Benchmarking Microsoft® .NET vs. Sun® Microsystem's J2EE: A Study of Performance and Scalability* at http://www.gotdotnet.com/compare for full details on the performance and scalability comparisons.

Introduction

The Java Pet Store is a reference implementation of a distributed application according the J2EE blueprints maintained by Sun Microsystems. The sample application is now in version 1.1.2 and was created by Sun to help developers and architects understand how to use and leverage J2EE technologies, and how J2EE platform components fit together. The Java Pet Store blueprint includes documentation, full source code for the Enterprise Java Beans™ (EJB) architecture, Java Server Pages (JSP) technology, tag libraries, and servlets that build the application. In addition, the Java Pet Store blueprint demonstrates certain models and design patterns through specific examples. The Java Pet Store is a best-practices reference application for J2EE, and is redistributed within the following J2EE-based application server products:

- IBM® Websphere® 4.0
- BEA® WebLogic® 6.1
- Oracle® 9i Application Server
- Sun Microsystems iPlanet®

The full Java Pet Store contains three sample applications:

1. Java Pet Store: The main J2EE Blueprints application.
2. Java Pet Store Administrator: The administrator module for the Java Pet Store.
3. Blueprints Mailer: A mini-application that presents some of the J2EE Blueprints design guidelines in a smaller package.

The original version of the Java Pet Store was designed to work with the following databases: Oracle, Sybase® and Cloudscape®. IBM has created a DB2 port of the application. The application is publicly available at the Java 2 Platform Enterprise Edition Blueprints website: http://java.sun.com/j2ee/blueprints/index.html.

The Java Pet Store application is designed as a J2EE "best-practices" application. The application provides an emphasis on the features and techniques used to show real world coding examples.

Functional Walkthrough

The premise of the original application, the Java Pet Store, is an e-commerce application where customers can buy pets online. When you start the application you can browse and search for various types of pets from dogs to reptiles.

A typical session using the Java Pet Store is as follows:

Homepage - This is the main page that loads when the user first starts the application.

Category View– There are 5 top-level categories: Each category has several products associated to it.

Products – When a product is selected within the application, all variants of the product are displayed. Typically the product variant is either male or female.

Product Details – Each product variant (represented as items) will have a detailed view that displays the product description, a product image, price, and the quantity in stock.

Shopping Cart – Allows the user to manipulate the shopping cart (add, remove, and update line items).

Checkout – The checkout page displays the shopping cart in a read-only view.

Login Redirect – When the user selects "Continue" on the checkout page, they are redirected to the login page if they have not signed in yet.

Verify Sign in – After being authenticated onto the site, the user is then redirected to the credit card and billing address form.

Confirm Order – The billing and the shipping addresses are displayed.

Commit Order – This is the final step in the order-processing pipeline. The order is now committed to the database at this point.

The Microsoft .NET Pet Shop

The goal of the .NET Pet Shop was to focus on the Java Pet Store itself (the administration and mailer component were not ported to .NET in this phase[4]). In addition to reproducing the functionality of the Java Pet Store application, two additional objectives were added:

1. Compare and contrast the architecture, programming logic and overall code size of this best-practice application between the .NET and J2EE implementations. This information is contained within this document.
2. Provide performance benchmark data on how well each application performs at varying user loads. This information is summarized in this document but discussed in detail in the document *Benchmarking Microsoft .NET vs. Sun Microsystem's J2EE: A Study of Performance and Scalability* (available at *http://www.gotdotnet.com/compare*).

A sample of the .NET Pet Shop can be seen in Figure 1.

[4] Note that in all code-count comparisons, we do a 1:1 code-count for only the functionality implemented, not counting lines of code for the Java administration or mailer applications.

Figure 1. The Microsoft .NET Pet Shop

The overall logical architecture of the .NET Pet Shop is detailed in Figure 2.

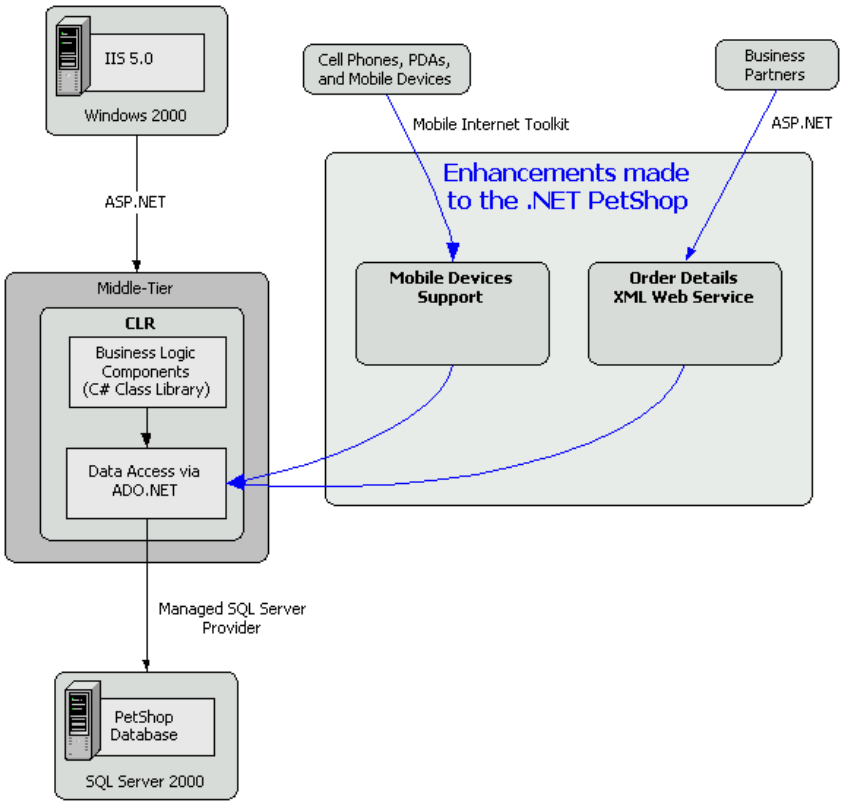


Figure 2. .NET Pet Shop Logical Architecture

There are three logical tiers: the presentation tier, the middle tier, and the data tier. The three tiers allow for clean separation of the different aspects of a distributed application. The business logic is encapsulated into a .NET Assembly (implemented as a C# Class Library). The database access is funneled through one class that handles all of the interaction with the SQL Server Managed provider. Access to the data stored in the database is accomplished through stored procedures. The application represents a complete logical three-tier implementation using .NET, and illustrates coding best-practices for the Microsoft.NET platform.

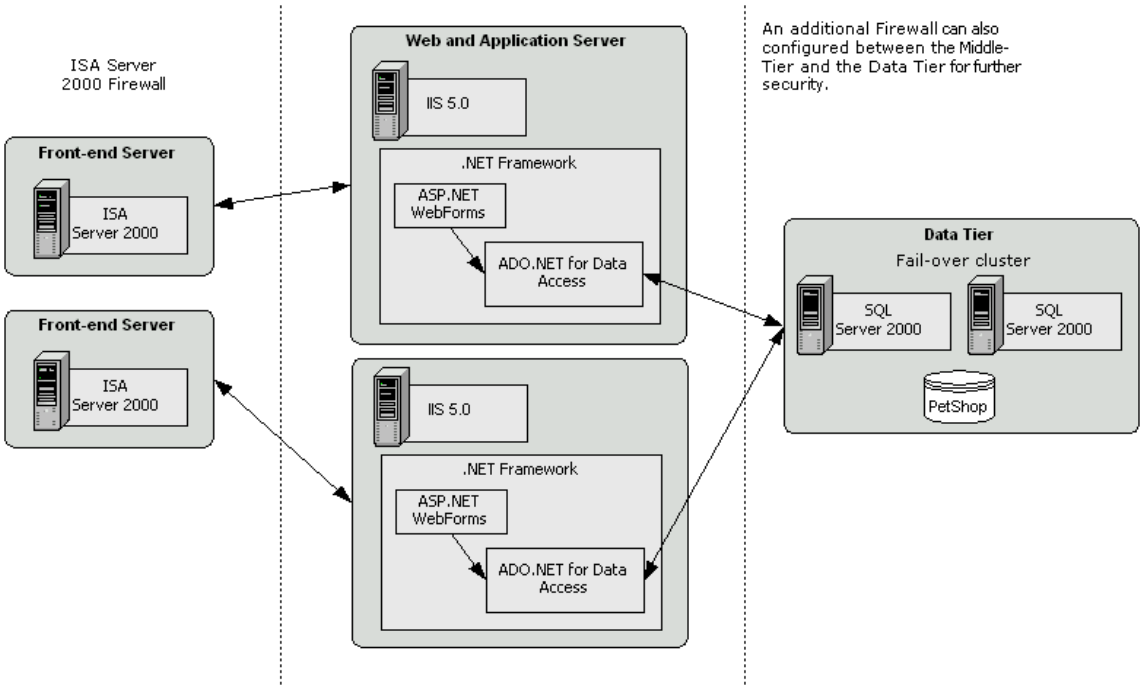


Figure 3. Sample .NET Pet Shop Physical Deployment Diagram

Architecture of the Application

The previous section addressed the high-level architecture of the application. To gain a better understanding of how the application works, this section will walkthrough a section of the code, demonstrating the interaction between the presentation tier, the middle-tier, and the data tier. To further illustrate the design, we will take a detailed look at the interaction and implementation details at each tier for the shopping cart as shown in Figure 4.

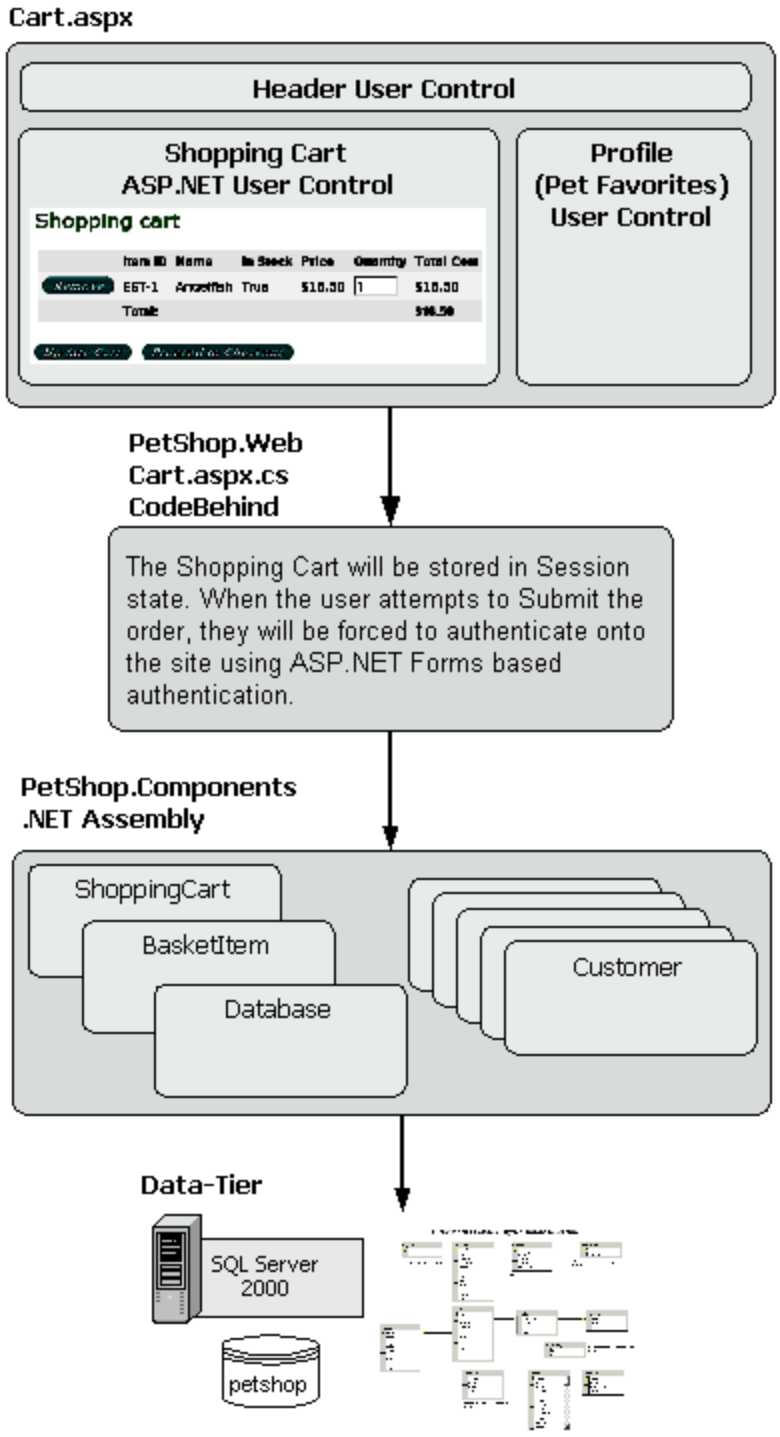


Figure 4. Architecture Walkthrough

In designing n-tier applications, there are a variety of approaches. In implementing the .NET Pet Shop, a data-driven approach was used.

Database

The database for the Java Pet Store is available in several database vendor formats. The first step we took was to port the Java Pet Store database schema to SQL Server 2000. This required no changes from the Sybase version of the schema. The database has the following overall structure of tables:

Table Name	Purpose
Account	Represents basic customer information.
BannerData	Stores the ad banner information.
Category	The catalog categories (i.e. Fish, Dogs, Cats, etc).
Inventory	Product inventory status.
Item	Individual product details.
LineItem	Order details.
Orders	The orders placed by the customer. An order contains 1 or more line items.
OrderStatus	Order status.
Product	The catalog products. Each product may have 1 or more variants (Items). A typical variant is usually male or female.
Profile	Customer user profile.
Signon	Login table for customers.
Supplier	Information concerning fulfillment suppliers.

Table 1. Database Table Names

The complete physical database schema for the .NET Pet Shop is illustrated in Figure 5.

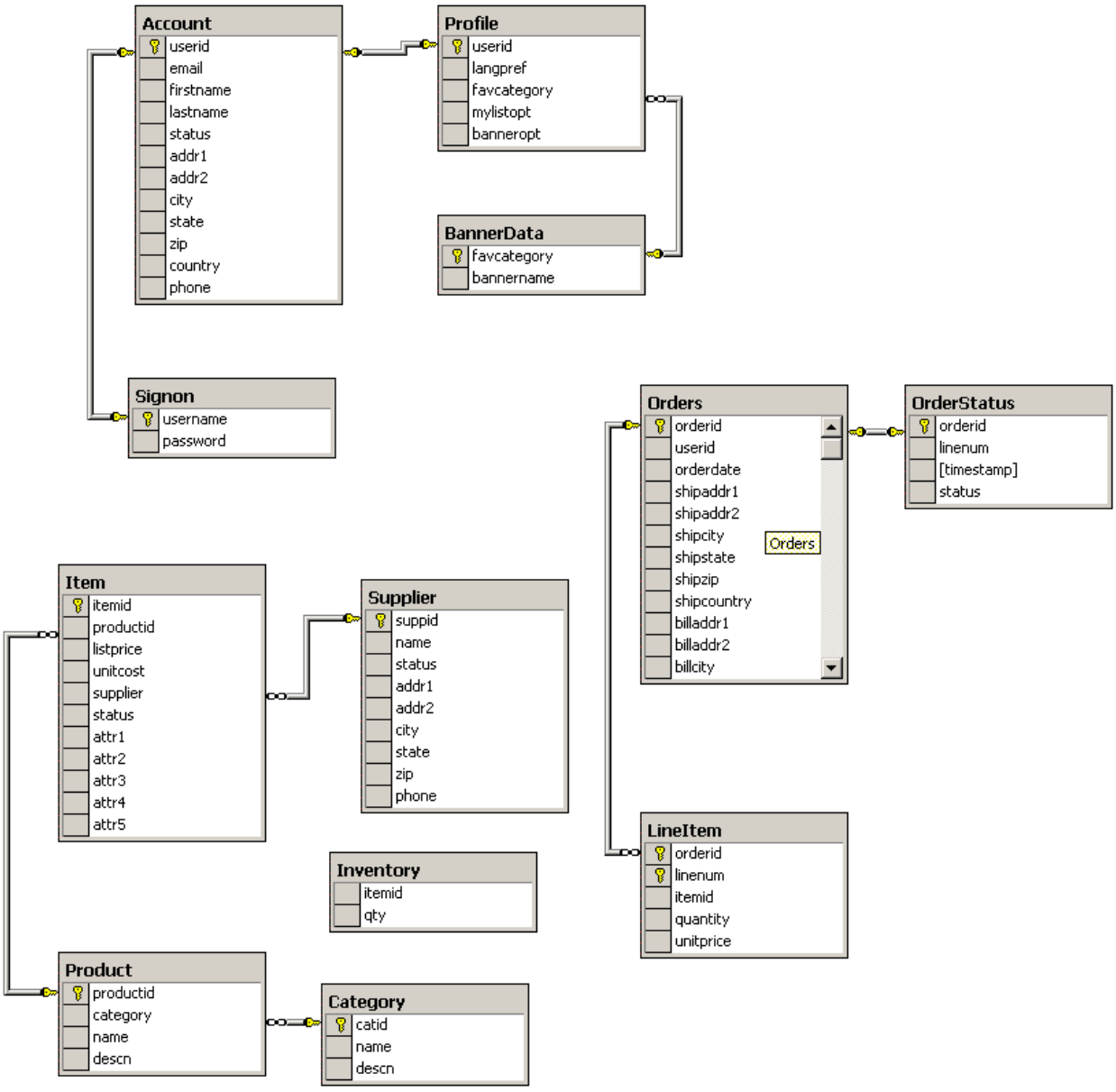


Figure 5. .NET Pet Shop Physical Database Schema

After careful consideration, some minor changes were made in terms of the relationships to improve the overall data integrity. While these changes were not required and do not add to the performance of the application, they added considerable benefits in clarifying the database design. The changes were:

- Added a foreign key relationship between the Signon and the Account table for the userid and username fields.
- Added a foreign key relationship between the Account and the Profile table for the userid and username fields.
- Added a foreign key relationship between the Profile and the BannerData table for the favcategory fields in each table.

- Removed the primary key constraint on the linenum, field in the OrderStatus table.
- 5 indexes have been added (4 on the Product table and 1 on the Item table).

The changes described above are shown in Figure 5.

The Sun Java Pet Store makes heavy use of Bean Managed Persistence (BMP) in its middle-tier Enterprise Java Beans (EJB). Essentially, this provides the objects a mechanism to persist their state to the database. The .NET Pet Shop takes a different approach in that the middle-tier components make calls to stored procedures in the database. Using stored procedures at the database level is well established as a best-practice for distributed applications. It provides a cleaner separation of code from the middle-tier and also helps clarify the transaction context and scope. Only basic queries are encapsulated in the stored procedures; business logic remains in the middle tier .NET classes.

Stored Procedure Name	**Purpose**
upAccountAdd	Adds an account to the customer database.
upAccountGetAddress	Adds a new address for a customer. This is used when the billing and shipping information are different.
upAccountGetDetails	Returns the details of a customer's account within the system.
upAccountLogin	Used to authenticate a user who is logging into the site.
upAccountUpdate	Updates the customer's account information.
upCategoryGetList	Returns a list of the categories in the catalog.
upInventoryAdd	Adds inventory information for a particular item.
upInventoryGetList	Gets the inventory status.
upItemAdd	Adds a product variant (referred to as an item) to the catalog.
upItemGetDetails	Gets the product information on a specific product variant.
upItemGetList	Returns the list of items (product variants) belonging to a particular product.
upItemGetList_ListByPage	Returns the list of items (product variants) belonging to a particular product. The data is returned in a user defined page size.
upOrdersAdd	Adds a customer order to the database. This stored procedure uses OpenXML and SQL Server transactions to execute its work.
upOrdersGet	Used to get a list of the orders by customer.
upOrdersGetDetails	Returns the full order details.
upOrderStatusGet	Returns the status of a particular order.

upProductAdd	Adds a product to the catalog.
upProductGetList	Returns the list of products in the catalog.
upProductGetList_ListByPage	Returns the list of products in the catalog. The data is returned in a user defined page size.
upProductSearch	Performs a product search.
upProductSearch_ListByPage	Performs a product search. The data is returned in a user defined page size.
upProfileGetBannerOption	Returns the customer's preference for displaying banner ads.
upProfileGetListOption	Returns the customer's preference for displaying the "Pet Favorites" list.

Table 2. Database Stored Procedures

In order to enhance the design of the application, the OpenXML feature of SQL Server 2000 was used to return XML documents instead of traditional rowsets. This was used heavily in the order checkout process because it simplifies the number of parameters that have to be managed between the middle-tier component and the database stored procedures.

Middle-Tier

Sun uses Enterprise Java Beans (EJBs) to implement the business logic of the application. Sun implemented the application using a mix of Entity and Session Beans to handle all of the logic.

EJB Name	EJB Type
Account	Entity Bean
Inventory	Entity Bean
Order	Entity Bean
ProfileMgr	Entity Bean
ShoppingCart	Stateful Session Bean
ShoppingClientController	Stateful Session Bean
AdminClientController	Stateless Session Bean
Catalog	Stateless Session Bean
Customer	Stateless Session Bean
Mailer	Stateless Session Bean

Table 3. Java Pet Store Business Logic

The .NET Pet Shop middle-tier business logic is encapsulated into a single .NET Assembly. The namespace is called Pet Shop.Components as illustrated in Figure 6.

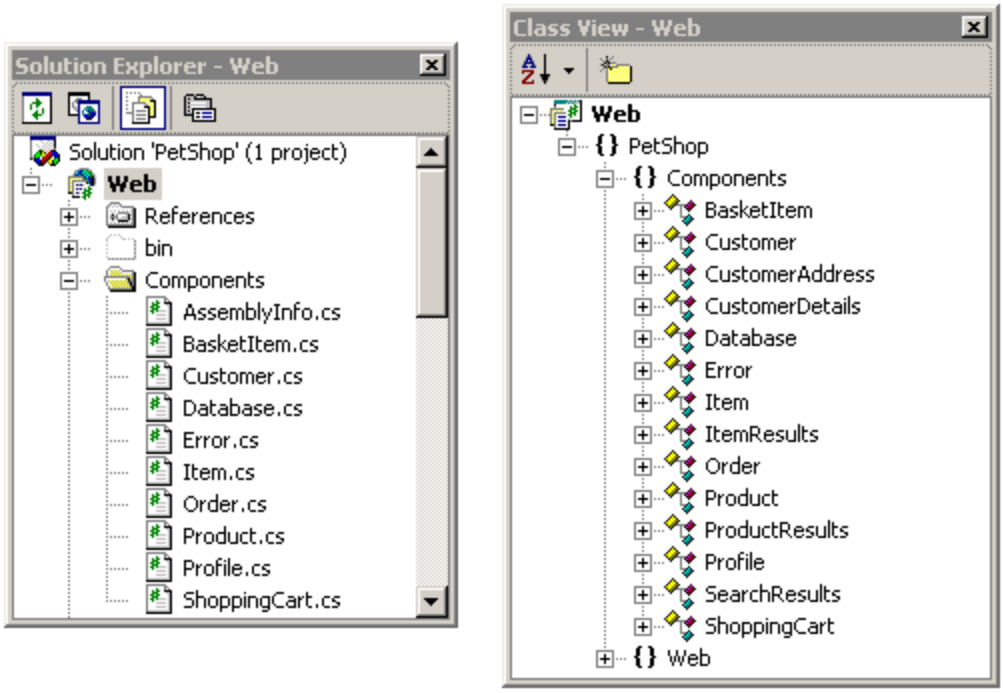


Figure 6. Middle-Tier Component Class View and Solution Explorer File View

There are 9 core classes, each one is implemented in its own .cs file.

Class Name	Purpose
BasketItem	A line item in the Shopping cart.
Customer	Account management and login verification.
Database	ADO.NET data access using the SQL Server Managed Provider.
Error	Helper functions for logging errors.
Item	Represents a product variant.
Order	Online orders transactions.
Product	A product in the catalog.
Profile	Used to update customer's profile information.
ShoppingCart	Shopping cart functions.

Table 4. Middle-Tier component

In addition there are 5 thin data classes: CustomerAddress, CustomerDetails, ItemResults, ProductResults, and SearchResults that are used to provide a louse coupling between the data-tier and the presentation data binding calls. Each of the thin data classes exposes public properties that can be data bound against in an ASP.NET WebForm.

```
/// <summary>
/// Contains the result of a catalog search.
/// </summary>
public class SearchResults {
      private string m_productid;
      private string m_name;
      private string m_descn;

      // search props
      public string productid {
            get { return m_productid; }
            set { m_productid = value; }
      }

      public string name {
            get { return m_name; }
            set { m_name = value; }
      }

      public string descn {
            get { return m_descn; }
            set { m_descn = value; }
      }
}
```

Code Snippet from the Order.cs illustrating the SearchResults thin data class.

All of the middle-tier code is compiled into a single assembly along with the code-behind logic in the presentation-tire in Pet Shop.dll.

For the code walkthrough, we will examine the shopping cart functionality for both Java and for .NET, starting first with the Java Pet Store.

The ShoppingCart EJB is a Stateful Session Bean, meaning the object maintains its contents for a specific amount of time on the server while the client is making a request. One common task of any shopping cart page is to add items to the user's basket. Before adding an item, the application must first determine which product is being added. In order to get the product information we will take a look at CatalogDAOImpl.java:

```
public Product getProduct(String productId, Locale locale) throws
       CatalogDAOSysException
{
        String qstr =
            "select productid, name, descn " +
            "from " +
            DatabaseNames.getTableName(DatabaseNames.PRODUCT_TABLE, locale) +
            " where " +
            "productid='" + productId + "'";
        Debug.println("Query String is:" + qstr);

        Product product = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            getDBConnection();
            stmt = dbConnection.createStatement();
```

```
            rs = stmt.executeQuery(qstr);
            while (rs.next()) {
                int i = 1;
                String productid = rs.getString(i++).trim();
                String name = rs.getString(i++);
                String descn = rs.getString(i++);
                product = new Product(productid, name, descn);
            }
        } catch(SQLException se) {
            throw new CatalogDAOSysException("SQLException while getting " +
                        "product " + productId + " : " + se.getMessage());
        } finally {
            closeResultSet(rs);
            closeStatement(stmt);
            closeConnection();
        }
        return product;
}
```

Code Snippet from the Java Petstore.

One thing to point out is that the middle-tier component contains a SQL statement inside of the class definition (see the qstr String object). The .NET Pet Shop takes a different approach and cleanly encapsulates all of the database logic into stored procedures.

```
public SqlDataReader GetList_ListByPage(string catid, int currentPage,
    int pageSize, ref int numSearchResults)
{
  // create data object and params
  SqlDataReader dataReader = null;
  try
  {
    // create params for stored procedure call
    Database data = new Database();
    SqlParameter[] prams = {
      data.MakeInParam("@cat_id", SqlDbType.Char, 10, catid),
      data.MakeInParam("@nCurrentPage", SqlDbType.Int, 4, currentPage),
      data.MakeInParam("@nPageSize", SqlDbType.Int, 4, pageSize),
      data.MakeOutParam("@totalNumResults", SqlDbType.Int, 4) };

    // run the stored procedure
    data.RunProc("upProductGetList_ListByPage", prams, out dataReader);
    numSearchResults = dataReader.RecordsAffected;
  }
  catch (Exception ex)
  {
    Error.Log(ex.ToString());
  }
  return dataReader;
}
```

Code Snippet for .NET Pet Shop

Notice that .NET version is making a call to RunProc(). This is a method in our reusable Database class and in this case will execute a stored procedure returning a SqlDataReader (very similar to a read-only, forward-only cursor in ADO):

```
public void RunProc(string procName, SqlParameter[] prams,
  out SqlDataReader dataReader)
{
 SqlCommand cmd = CreateCommand(procName, prams);
 dataReader = cmd.ExecuteReader(System.Data.CommandBehavior.CloseConnection);
}
```

Code Snippet from .NET Pet Shop Data Tier

The snippets of code for the .NET Pet Shop is essentially the skeleton for all data access in the application. The Java application data access mechanisms vary from EJB to EJB. There are some sessions objects, but for the most part the components are bean-managed, meaning that the object itself is responsible for persisting itself to the database, hence the inline SQL code.

The .NET Pet Shop is using ASP.NET Session state to store the contents of the shopping cart (matching the functionality of the Java Pet Store). The session state is stored in-process, but ASP.NET also includes the ability to run a dedicated State Server and supports SQL Server database session state as well.

Presentation-Tier

The presentation-tier for the Pet Shop was written using ASP.NET Web Forms combined with User Controls. The site was created with Visual Studio .NET and therefore uses code-behind where the code for each ASPX page is encapsulated into a separate file. The Java Pet Store makes heavy use of Servlets (or mini java-server programs that emit HTML). The Servlets are using a design pattern called the Model View Controller, allowing the data to be displayed and manipulated in several different ways. The .NET Pet Shop accomplishes this same functionality using the ASP.NET Server Controls in addition to the User Controls.

Since the .NET Pet Shop application makes uses of Server Controls and Session State, careful consideration was given to their parameters to achieve high performance. The configuration is detailed in Table 5.

ASP.NET WebForms	EnableSessionState	EnableViewState
Cart.aspx	true	true
Category.aspx	true	false
CheckOut.aspx	readonly	false
CreateNewAccount.aspx	false	true
Default.aspx	false	false
EditAccount.aspx	false	true
Error.aspx	false	false
Help.aspx	false	false
OrderAddressConfirm.aspx	readonly	false
OrderBilling.aspx	true	true
OrderProcess.aspx	readonly	false

OrderShipping.aspx	true	true
Product.aspx	false	false
ProductDetails.aspx	false	false
Search.aspx	false	false
SignIn.aspx	false	false
SignOut.aspx	true	false
ValidateAccount.aspx	false	false
VerifySignIn.aspx	false	false

Table 5. WebForm Session State and View State settings. These settings are included as 1-line directives at the top of the individual aspx files.

ASP.NET Output Caching

One of best ways to improve the performance of a database driven application is to remove hitting the database for every query. ASP.NET offers a variety of caching mechanism that add considerable performance benefits to the application. Output Caching is used to take the contents of a page and store the results in memory. Subsequent requests for that page are served from the cache and do not require accessing the database again. The Java Pet Store as implemented by Sun does not have a way to do this very easily. While various J2EE application servers have output cache capabilities (for example, IBM Websphere 4.0), these features vary from product to product. Using these features may require modifying the Java Pet Store source code to work with a given J2EE application server.

The .NET Pet Shop application also uses Partial Page (Fragment) Caching to cache various regions of page. For instance, the header information that appears on the top of every page is cached. However, the header information is dependent on whether or not the user has signed in (so we technically have to cache two different versions of the page). ASP.NET easily allows this by using the VaryByCustom attribute on the OutputCache directive. Using VaryByCustom simply requires overriding the GetVaryByCustomString method to get a custom cache for the header.

```
/// <summary>
/// Custom caching using the VaryByCustom attribute in the
/// OuputCache directive.
/// </summary>
/// <param name="context">The current HTTP request</param>
/// <param name="arg">Custom cahce argument</param>
/// <returns>Cache key</returns>
public override string GetVaryByCustomString(HttpContext context, String arg)
{
      // cache key that is returned
      string cacheKey = "";

      switch(arg) {
            // Custom caching for header control. We want to create two views
            // of the header... one if the user is logged in and another if
            // they are logged out.
            case "CategoryPageKey":
                  if (Request.IsAuthenticated == true) {
                    cacheKey = "QQQ" +
```

```
                    context.Request.QueryString["category_id"] +
                    context.Request.QueryString["requestedPage"];
                }
                else {
                  cacheKey = "AAA" +
                    context.Request.QueryString["category_id"] +
                    context.Request.QueryString["requestedPage"];
                }
                break;
                                .
                                .
                                .
                case "UserID" :

                     if (Request.IsAuthenticated == true) {
                           cacheKey = "UserID_In";
                     }
                     else {
                           cacheKey = "UserID_Out";
                     }

                     break;
     }

     return cacheKey;
}
```

ASP.NET Code Snippet

All of the cache settings are summarized in Table 6.

ASP.NET WebForms	Cache setting	Duration	Notes
ControlHeader	<%@ OutputCache Duration="43200" VaryByParam="none" VaryByCustom="UserID" %>	12 hours	Uses custom cache that calls GetVaryByCustomString. This returns one of two cache keys… one if the user is logged in and another if they are logged out.
Default	<%@ OutputCache Duration="43200" VaryByParam="none" VaryByCustom="UserID" %>	12 hours	Static content on main page. Could be cached for longer.
Help	<%@ OutputCache Duration="43200" VaryByParam="none" VaryByCustom="UserID" %>	12 hours	Static content on help page. Could be cached for longer.
Category	<%@ OutputCache Duration="43200" VaryByParam="none" VaryByCustom="CategoryPageKey " %>	12 hours	Different cache for each category.
Product	<%@ OutputCache Duration="43200" VaryByParam="none" VaryByCustom="ProductPageKey " %>	12 hours	Different cache for each product.
ProductDetails	<%@ OutputCache Duration="43200"	12 hours	The duration is low since the item quantity is displayed.

	VaryByParam="none" VaryByCustom="ProductDetailsPageKey " %>		
Search	<%@ OutputCache Duration="43200" VaryByParam="none" VaryByCustom="SearchPageKey " %>	12 hours	Very useful since caches popular queries and don't have to keep hitting the database when someone searches for cat.

Table 6. .NET Pet Shop Cache Settings

As mentioned previously, in addition to Output Caching the WebForms, the .NET Pet Shop also takes advantage of Fragment Caching which allows a region of page to be cached. The header ASP.NET User Control that appears on every page is cached.

ASP.NET User Controls	EnableViewState	Output Caching
Control_Address.ascx	true	no
Control_Banner.ascx	true	no
Control_Cart.ascx	true	no
Control_FavList.ascx	true	no
Control_Header.ascx	false	yes, VaryByCustom
Control_StaticAddress.ascx	true	no

Table 7. User Control View State and Output Cache settings

Error Handling

The error handling in the .NET PetShop was been simplified by using custom errors for ASP.NET. All thrown exceptions are now being caught in the Application_Error() method in the Globabl.asax.cs file.

```
/// <summary>
/// Global application level error handling. This method is invoked
/// anytime an exception is thrown.
/// </summary>
/// <param name="sender">Sender</param>
/// <param name="e">Event arguments</param>
private void Application_Error(object sender, System.EventArgs e) {
      string err = "Microsoft .NET PetShop Application Error\n\n";
      err += "Request: " + Request.Url.ToString() + "\n\n";
      err += "Strack Trace: \n" + Server.GetLastError().ToString();

      // write the error message to the NT Event Log
      Components.Error.Log(err);
}
```

This allows the PetShop to have a global application-level error handling. If an error occurs, the user is directed to a default error page (Error.aspx). The full stack trace and requested URL that generated the error is then written to the Application Event Log for the system administration, prevent the end user from seeing an unhanded exception error in the browser.

Security

Key components in any application are the security considerations of the application. Both the Java Pet Store and the .NET Pet Shop leverage a variety of security mechanism to prevent break-ins and attacks. The .NET Pet Shop uses ASP.NET Forms-based Authentication. These settings are centralized in the web.config file:

```
<!--   AUTHENTICATION
       This section sets the authentication policies of the application.
       Possible modes are "Windows", "Forms", "Passport" and "None"
-->
<authentication mode="Forms">
    <forms name="Pet ShopAuth"
           loginUrl="SignIn.aspx"
           protection="All"
           path="/" />
</authentication>
```

The Java Pet Store exhibits similar functionality, but security meta data is expressed in Web component deployment descriptors:

```
<web-app>
  <login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>login.jsp</form-login-page>
      <form-error-page>login.jsp</form-error-page>
    </form-login-config>
  <login-config>
<web-app>
```

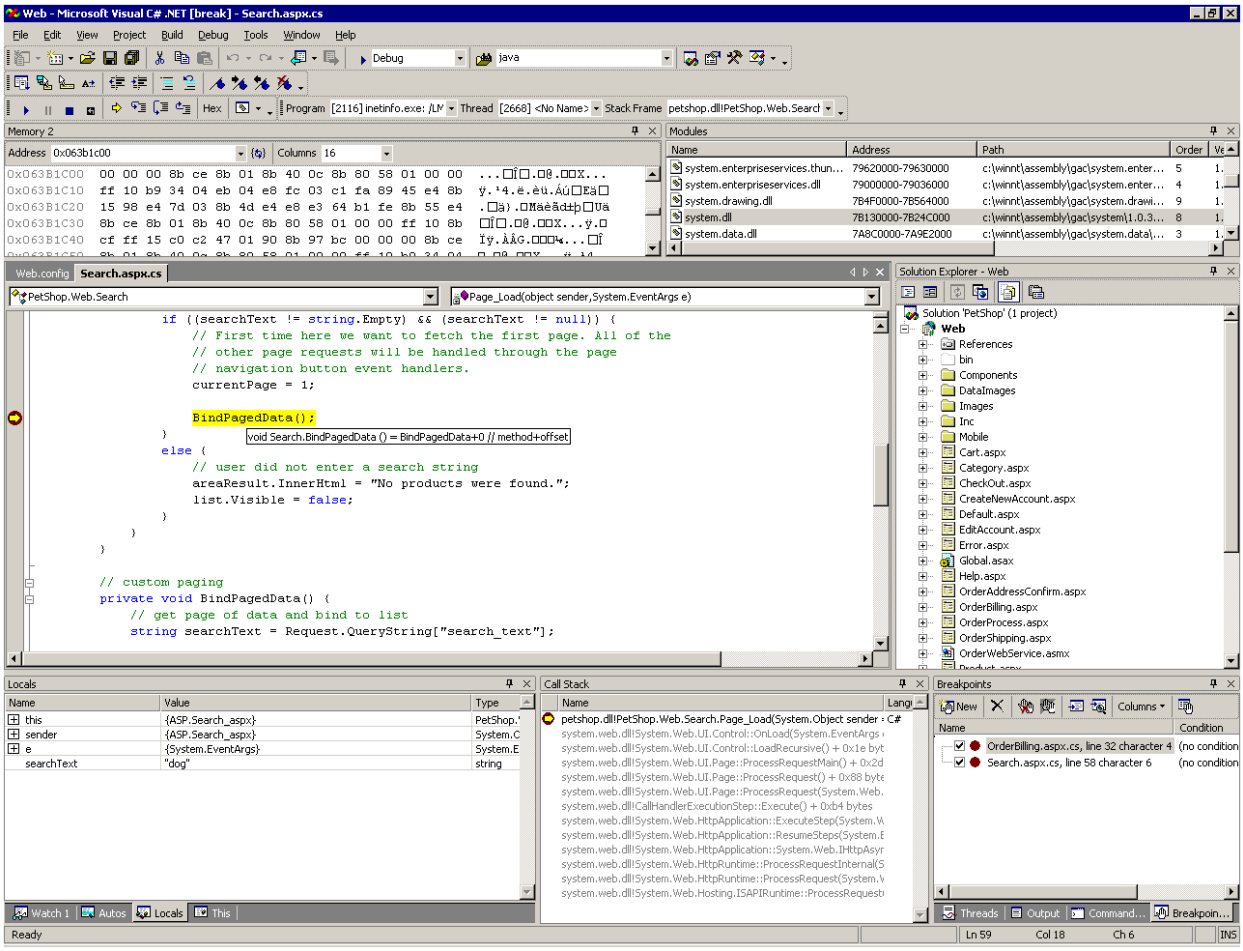
The security behavior of the Java Pet Store and the .NET Pet Shop are the same: ASP.NET allows all of the security configuration to be centralized into one configuration file while the Java version has several configuration files.

Debugging

A key step in developing any type of software application is debugging. During the development of the .NET PetShop, we discovered bugs easily with the Visual Studio .NET integrated debugging. We were able to debug all three tiers from the presentation tier down to the stored procedures in SQL Server within one session of Visual Studio .NET. Before we can debug, we needed to ensure that our web.config had the debug attribute in the compilation element set to true.

```
<compilation defaultLanguage="c#" debug="true" />
```

Here is a typical debugging session for the Search.aspx WebForm. We have set a breakpoint right before we are going to do our data binding. You can also note that the input search text in this session was dog (see the lower left hand side of the screenshot).


Web - Microsoft Visual C# .NET [break] - Search.aspx.cs
File Edit View Project Build Debug Tools Window Help
Debug
java
Memory 2
Modules
Name Address Path Order
system.enterpriseservices.dll 79000000-79036000 c:\winnt\assembly\gac\system.enterprise... 4
system.drawing.dll 7B4F0000-7B564000 c:\winnt\assembly\gac\system.drawing... 9
system.dll 7B130000-7B24C000 c:\winnt\assembly\gac\system\1.0.3... 8
system.data.dll 7A8C0000-7A9E2000 c:\winnt\assembly\gac\system.data\... 3
Web.config
Search.aspx.cs
PetShop.Web.Search
Page_Load(object sender,System.EventArgs e)
if ((searchText != string.Empty) && (searchText != null)) {
// First time here we want to fetch the first page. All of the
// other page requests will be handled through the page
// navigation button event handlers.
currentPage = 1;
BindPagedData();
void Search.BindPagedData () = BindPagedData+0 // method+offset
else {
// user did not enter a search string
areaResult.InnerHtml = "No products were found.";
list.Visible = false;
// custom paging
private void BindPagedData() {
// get page of data and bind to list
string searchText = Request.QueryString["search_text"];
Solution Explorer - Web
Solution 'PetShop' (1 project)
Web
References
bin
Components
DataImages
Images
Inc
Mobile
Cart.aspx
Category.aspx
CheckOut.aspx
CreateNewAccount.aspx
Default.aspx
EditAccount.aspx
Error.aspx
Global.asax
Help.aspx
OrderAddressConfirm.aspx
OrderBilling.aspx
OrderProcess.aspx
OrderShipping.aspx
OrderWebService.asmx
Locals
Name Value Type
this {ASP.Search_aspx} PetShop.
sender {ASP.Search_aspx} System.C
e {System.EventArgs} System.E
searchText "dog" string
Call Stack
Name
petshop.dll!PetShop.Web.Search.Page_Load(System.Object sender
Breakpoints
New
Columns
Name Condition
OrderBilling.aspx.cs, line 32 character 4 (no condition
Search.aspx.cs, line 58 character 6 (no condition
Watch 1 Autos Locals This
Threads Output Command... Breakpoin...
Ready
Ln 59 Col 18 Ch 6
INS

Comparing the Code Size

It is interesting to compare the overall lines of implementation code in the two versions of the application to get an understanding of the relative developer productivity of Microsoft .NET vs. Sun's J2EE development platform. A comprehensive code-count reveals that the lines of code required to implement the Java Pet Store is over 3.5 times the amount of code required to implement the same functionality in the .NET Pet Shop. The following tables give a more complete break down by tier.

	Microsoft .NET Pet Shop		**Sun Java Pet Store**[5]		**Overall Comparison**
	Lines of Code	**# Files**	**Lines of Code**	**# Files**	
Presentation	1,881	52	5,891	131	J2EE version has three times the amount of UI code as .NET version
Middle-Tier	863	10	5,404	121	J2EE version has 6.2 times the amount of middle tier code as the .NET version
Database	684	2	412	1	J2EE version has 60% the amount of data tier code as the .NET version
Config	56	1	2,566	19	J2EE version has 45.6 times the amount of configuration code as the .NET version
Total	**3,484**	**65**	**14,273**	**273**	**J2EE has 4 times the amount of total code as the .NET version**

Table 8. Lines of code broken down by tier.

[5] The code-count of the Java Pet Store includes only the modules that were ported to .NET.

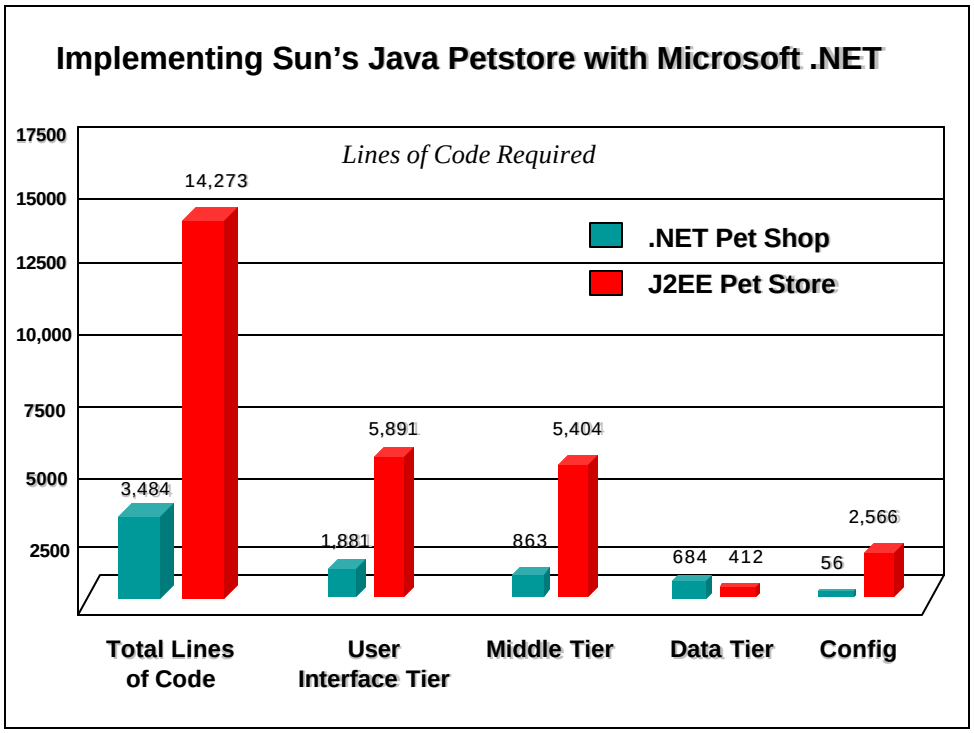


Figure 7: Lines of code broken down by tier.

	.NET Summary	Java Summary
Lines of Code	3,484	14,273
Number of Files	65	273
Folder Count	7	110
Size (in bytes)	438,272	811,094

Table 9. Other comparisons

The mechanism used to perform the code count is detailed in Appendix 1.

Why the .NET code base is much smaller and more efficient

The Java Pet Store contains several complex design patterns that result in making the J2EE blueprint application very challenging for developers to reuse. The Java Pet Store takes a complete object-oriented approach which results in very heavy middle-tier objects. The use of the J2EE-recommended bean managed persistence and other EJB functionality adds to the complexity of the design. The .NET Pet Shop, on the other hand, takes a streamlined component approach with very small, light-weight objects that take advantage of several .NET features that significantly reduce the overall code size and increase developer productivity. For example, the use of ASP.NET Server Controls and ASP.NET Web Forms help reduce the server-side and client-side programming burden on developers significantly. Also, the Java code also has to

manually update pages using its Model View Controller design pattern where the .NET code can simply use DataBinding with ASP.NET Server Controls like the DataGrid.

On the data tier, the core database schema (the tables and all of the relationships and constraints) are virtually identical for both versions of the application. In both, the schema is roughly 400 lines of SQL code. The .NET version has slightly more lines of code because we added 3 additional foreign key constraints to ensure data integrity.

However, the Data-Tier for the .NET Pet Shop is larger than the Java Pet Store for the following reasons: The .NET Pet Shop is using Stored Procedures to encapsulate all of the database insert, updates, and deletes. The Java Pet Store EJBs are using bean-managed persistence to handle the insert, updates, and deletes. This means that the Java Pet Store has SQL statements in the Middle-Tier code, not the Data-Tier. So the functionality for Java is being counted in the Middle-Tier category versus the Data-Tier category. We believe the .NET implementation with stored procedures is not only a superior design pattern, but also offers much better performance and scalability over the J2EE bean-managed persistence model. The benchmarks of the two versions of the application validate this point.

Finally, there are 3 stored procedures to support the "additional" Web Service and Mobile features. The Java Pet Store does not have these features. We included these in the code count of the Data Tier of the .NET Pet Shop application so that customers downloading the .NET code could perform their own code count as-is without having to factor out any features.

Performance and Scalability Comparison

The .NET Pet Shop was benchmarked and compared to an optimized and highly tuned version of the Java Pet Store as benchmarked by Oracle Corporation. The results of the overall performance of the application are summarized here, but fully described in *Benchmarking Microsoft .NET vs. Sun Microsystem's J2EE: A Study of Performance and Scalability* (available at http://www.gotdotnet.com/compare). All results shown here for J2EE are cited from Oracle's publicly published data included in their whitepaper *Oracle9iAS J2EE Performance Study Results*. The Oracle Java Pet Store benchmark tests were precisely replicated using the .NET Pet Shop on the same hardware configuration which included a 2-CPU middle tier application server and a 4-CPU database server.

	Per page user response times at 450 concurrent users	**App Server CPU Utilization % at 450 concurrent users**	**User Load supported at 1 second avg. response time**
.NET withASP.NET Output Caching	Microsoft .NET Pet Shop 28 times faster than Java Pet Store	Java Pet Store requires 6 times more CPU utilization	.NET Pet Shop supports 7.6 times more concurrent users than Java Pet Store
.NET with no ASP.NET Output Caching	.NET Pet Shop 15 times faster than Java Pet Store	Java Pet Store requires 4.3 times more CPU utilization	.NET Pet Shop supports 6.6 times more concurrent users than Java Pet Store

New Features

While porting the Java Pet Store to .NET, we found that the feature list of the application could easily be extended without much development time and code to include an XML Web Service and support for mobile devices.

XML Web Services

Using Visual Studio.NET, we quickly added an XML Web Service (based on the SOAP and UDDI standards) that returns the order details given an order id. For a complete document describing how to build this Web Service in Visual Studio.NET, including a direct comparison to building it with IBM WebSphere 4.0, please see the document titled *Building XML-based Web Services in Microsoft .Net vs. IBM Websphere 4.0*, available at *http://www.gotdotnet.com/compare*.

The sample XML output of the GetOrderDetails() SOAP method in OrderWebService.asmx Web Service is shown in Figure .

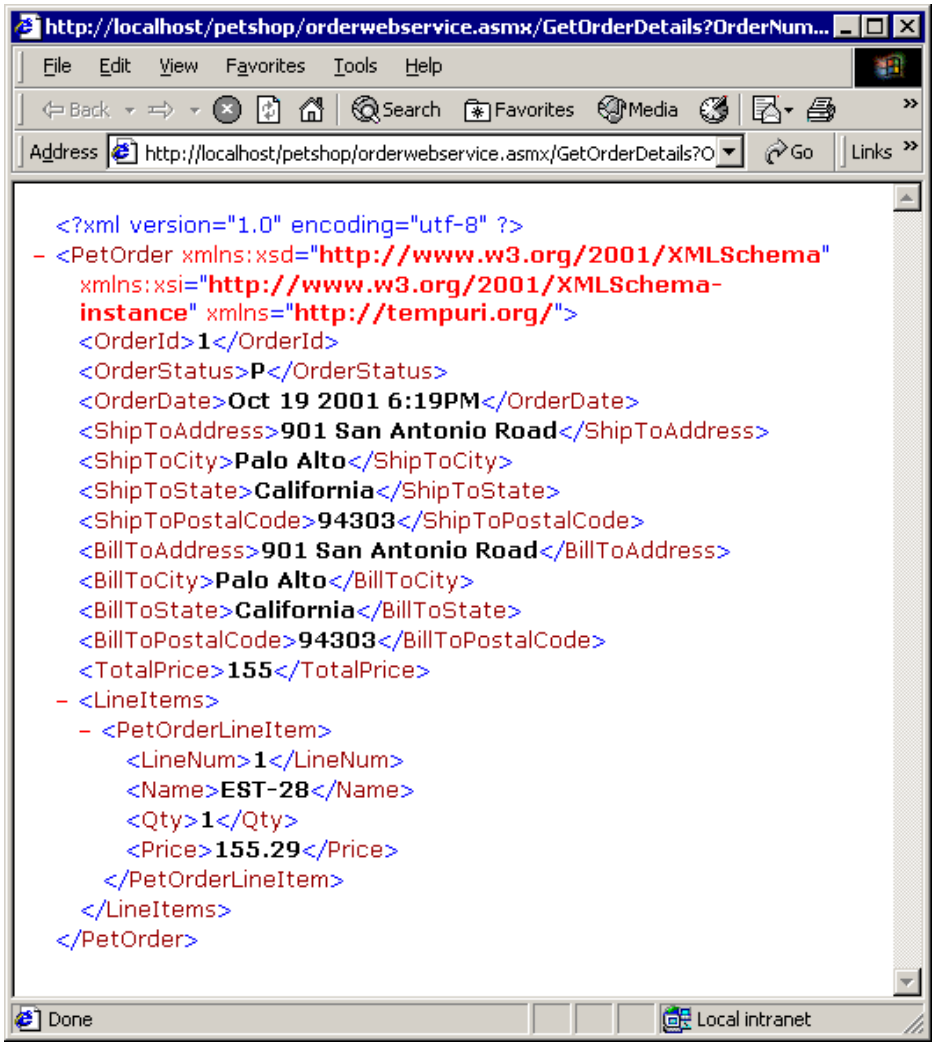


Figure 8. GetOrderDetails XML Web Service sample output

Mobile Device Support

ASP.NET provides some excellent mechanisms to target a variety of web browsers ranging from previous versions of Internet Explorer to Netscape. One area of growth for web applications has been extending to browsing with a variety of devices ranging form PocketPCs (Windows CE) running Pocket Internet Explorer to cell phones using WAP browsers. The Microsoft Mobile Internet Toolkit provides a set of assemblies that allow developers to write one code base that can support a multitude of different devices. During the port to .NET, we decided that it would be great if customers could check the status of their pet orders using a cell phone.

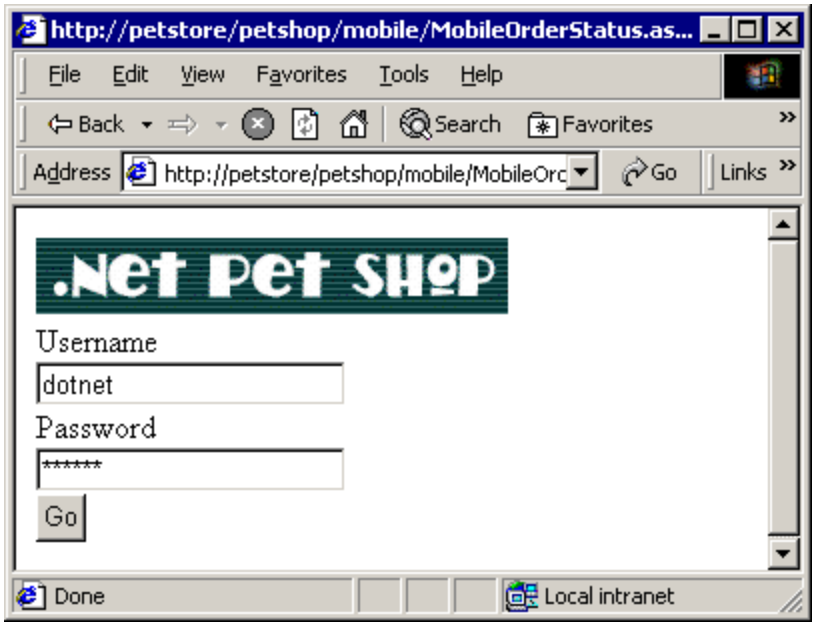


Figure 9. Mobile Order Status Page

Using the Mobile Internet Toolkit was very straight-forward. After running the setup, we had a new namespace called System.Web.Mobile (System.Web.Mobile.dll). The toolkit also integrates nicely into Visual Studio .NET making it possible to create the page using the visual design surface as shown in Figure 10. Developing MobileOrderStatus.aspx with Visual Studio .NET

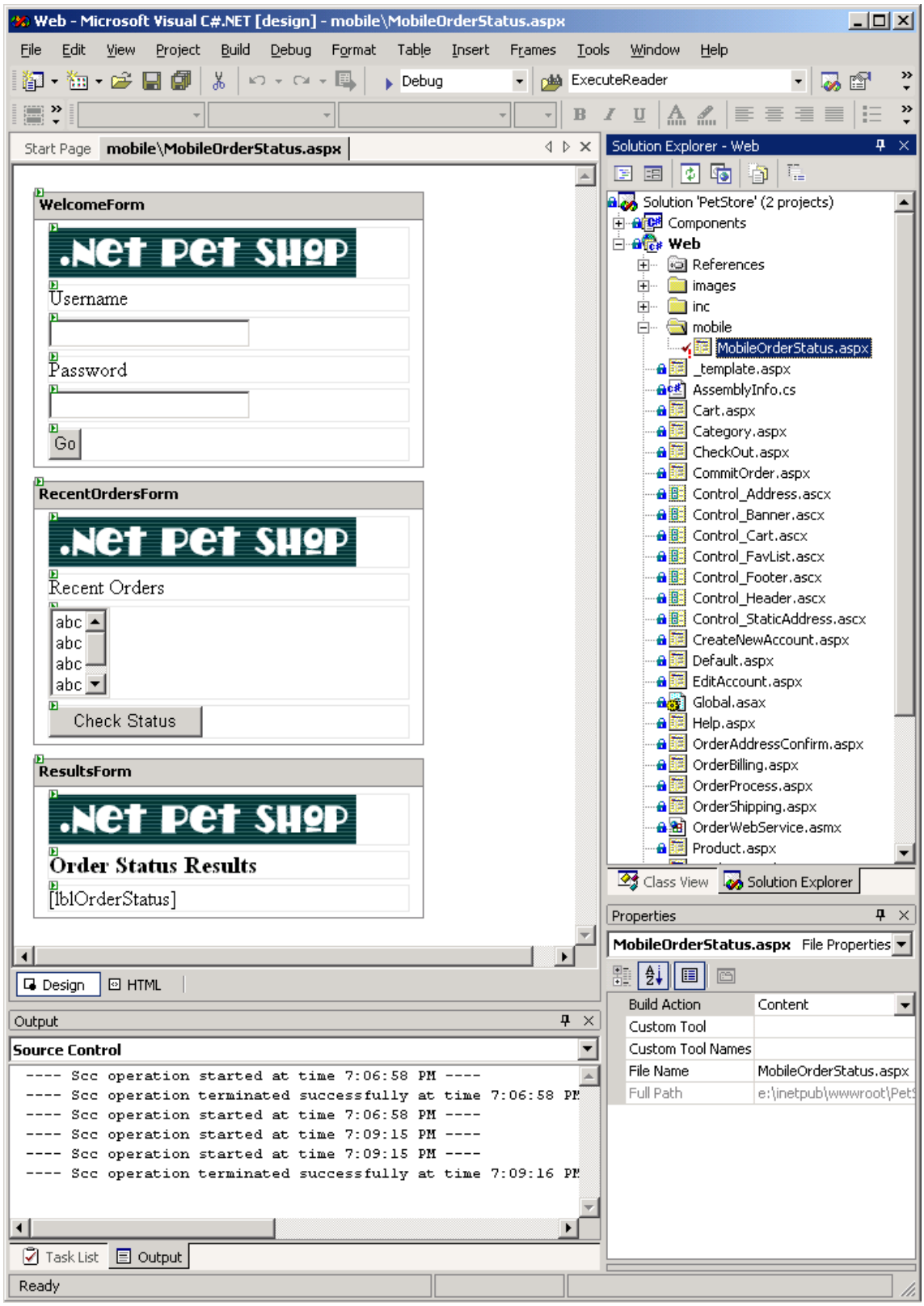


Figure 10. Developing MobileOrderStatus.aspx with Visual Studio .NET

The code for the mobile form can be seen below:

```
<body xmlns:mobile="Mobile Web Form Controls">

  <mobile:form id="WelcomeForm" runat="server">
      <mobile:Image id="imgLogoWelcome" runat="server"
                    AlternateText="Pet Shop"
                    ImageURL="../images/mobile_title.gif"></mobile:Image>
      <mobile:Label id="lblUsername" runat="server">Username</mobile:Label>
      <mobile:TextBox id="txtUsername" runat="server">dotnet</mobile:TextBox>
      <mobile:Label id="lblPassword" runat="server">Password</mobile:Label>
      <mobile:TextBox id="txtPassword" runat="server"
                      Password="True"></mobile:TextBox>
      <mobile:Command id="btnGo" runat="server">Go</mobile:Command>
  </mobile:form>

  <mobile:form id="RecentOrdersForm" runat="server">
      <mobile:Image id="imgLogoRecentOrders" runat="server"
                    AlternateText="Pet Shop"
                    ImageURL="../images/mobile_title.gif"></mobile:Image>
      <mobile:Label id="lblRecentOrders" runat="server">
         Recent Orders
      </mobile:Label>
      <mobile:SelectionList id="lstRecentOrder" runat="server"
                            SelectType="ListBox">
      </mobile:SelectionList>
      <mobile:Command id="btnCheckStatus" runat="server">
         Check Status
      </mobile:Command>
  </mobile:form>

  <mobile:form id="ResultsForm" runat="server" StyleReference="subcommand">
      <mobile:Image id="imgLogoResults" runat="server"
                    AlternateText="Pet Shop"
                    ImageURL="../images/mobile_title.gif"></mobile:Image>
      <mobile:Label id="lblResults" runat="server">
         Order Status Results
      </mobile:Label>
      <mobile:Label id="lblOrderStatus" runat="server"
                    Font-Size="Normal"></mobile:Label>
  </mobile:form>

</body>
```

The code-behind file MobileOrderStatus.aspx.cs contains the logic this page in the two button click event handlers. The first button “Go” will verify the user’s credentials and then display the list of orders belonging to that customer.

```
// go command event handler
private void btnGo_Click(object sender, System.EventArgs e)
{
      // verify login
      Customer customer = new Customer();
      if (customer.Login(txtUsername.Text, txtPassword.Text) != null)
      {
            // get the list of orders for the specified customer
            Order order = new Order();
            lstRecentOrder.DataValueField = "orderid";

            lstRecentOrder.DataSource = order.GetOrder(txtUsername.Text);
            lstRecentOrder.DataBind();
            lstRecentOrder.SelectedIndex = 0;
```

```
            // advance to the next card
            ActiveForm = RecentOrdersForm;
      }
}
```

The last button will check the selected order's status in the system.

```
// order status command event handler
private void btnCheckStatus_Click(object sender, System.EventArgs e)
{
      // get the order status
      Order order = new Order();
      lblOrderStatus.Text =
          order.GetOrderStatus(lstRecentOrder.Selection.Text);

      // advance to the results card
      ActiveForm = ResultsForm;
}
```

The benefit of using the Mobile Internet Kit is realized whenever the developer will need to support a wide range of devices and browsers with a variety of capabilities. To test cell phone support for the mobile page, a great emulator can be found on the developer section Openwave.com. We tested this mobile page with the Openwave SDK version 4.1 which can be downloaded at http://developer.openwave.com/download/index.html.

Conclusion

The Java Pet Store is Sun's reference blueprint application for building enterprise Web applications with the J2EE technology. The .NET Pet Shop is a blueprint application that serves to highlight the key Microsoft .NET technologies and architecture that can also be used to build enterprise Web applications. Now architects and developers can compare .NET and J2EE side by side using functionally identical reference applications that illustrate best coding practices for each platform.

Appendix 1: Counting the Lines of Code

We were very careful to count Java Pet Store code only for the precise functionality implemented in the .NET Pet Shop application. Lines of code for the Administration functionality and the Blueprint Mailer application were not counted, since this functionality was not included in the .NET Pet Shop. In addition, we factored out code in the Java Pet Store for supporting other databases beyond Oracle. To come up with line counts for both the Java and .NET versions of the code, we created a program that looks only at those lines considered important in each file. For instance, to count the lines of code in a file containing C# or Java code, we wanted to ignore blank lines and lines containing comments. Table 10 describes in more detail what lines where counted in which types of files.

File type	Which lines were counted
.cs	Any line containing C# code
.java	Any line containing Java code
.js	Any line containing JavaScript code
.asax, .aspx	Any line containing server-side code (C#)
.jsp	Any line containing server-side code (Java)
.sql	Any line containing SQL code

Table 10. Lines of code counted

In addition, for .asax, aspx, and .jsp files, if more than one server-side block (delimited by "<%" and "%>") appeared on the same line, each was counted as a separate line. For all other file types (e.g. .xml), all lines were counted. The line counting program was written as a lexical specification for **lex**, a readily-available lexical analyzer and code generator. In our case, we used **flex**, the GNU equivalent whose source code is freely available from the Free Software Foundation. Either can be used to produce a C source file from the specification file. The C file can, in turn, be compiled into an executable. A makefile for VC6 and VC7 is included in the distribution.

The usage of the program is quite simple. It accepts zero, one, or two command line arguments, depending on what kind of file is to be counted and whether the file to be counted is specified on the command line or piped in from another source. Table 11 describes the accepted command line arguments and how they are intended to be used.

Argument	Effect	Intended file types
(none)	Assumes file contains a mixture of HTML and server-side code	.asax, .aspx, .jsp
-all	Counts all lines	All others (e.g. .html, .xml, and .txt)
-code	Counts only lines of code	.cs, .java, .js
-sql	Counts only lines of SQL code	.sql

Table 11. cloc command line arguments

Since the line counting program accepts no more than one file at a time, we also created a collection of batch files to help count all of the lines in all of the directories for both the Java and .NET versions. These files are also included in the distribution.

In our makefile, we decided to have **flex** produce a C source file with a **.cpp** extension. Since VC compiles such files as C++, this allowed us to use the **bool** type and gave us better type safety. However, it also expects an include file called **unistd.h**. This is a standard include file on many systems but is not part of Visual C++. However, this file does not contain any declarations relevant to our line counting tool, so using an empty file is fine.

If you would like to reproduce our line count results, you may run the batch files yourself. The first one, **Extensions Used.bat**, produces a listing of the file extensions used in a directory hierarchy. The path of the top-level directory of the directory hierarchy is specified as the only command line argument. Files without extensions do not show up in the listing, but there are no files without extensions we needed to count. We ran this batch file on the Java version and the .NET version to determine which file extensions each used. Table 12 shows the file extensions found in each version, as well as which ones we used to compute the total line counts for each.

Platform	Extensions counted	Extensions not counted
Java	java, js, jsp, sql, tld, txt, xml	bat, html, mf, sh
.NET	asax, ascx, aspx, config, cs, sql, txt	bat, cmd, css, html, gif, jpg, resx, vbs, vsdisco

Table 12. Extensions

The other two batch files, Count Java Lines.bat and Count .NET Lines.bat, count the number of lines in the files in the Java and .NET versions, respectively. These are separate batch files since the extensions those batch files looks for are different, as shown in Table 12. The usage of these two batch files is the same. They both take a single command line argument that is the path of the top-level directory of the directory hierarchy to search for files to count.

To get the line counts for the different tiers, we ran the batch files on specific subdirectories and only included specific extensions, as shown in Table 13.

Platform		Presentation	Logic	Database	Configuration
Java	Extensions	java, js, jsp, txt, xml	java, txt, xml	sql	tld
	Subdirectories	Web	Components	Database	<All>
.NET	Extensions	asax, ascx, aspx, cs, css	cs	sql	config
	Subdirectories	Pet Store	components	src	<All>

Table 13. Final code breakdown

Determining a satisfactory line count across different file types is difficult enough without also attempting to count across different platforms. For instance, we did not count HTML files because the tools used to generate them may format the code differently on different platforms, thus producing different line counts. Although this also applies to how the logic tier files are formatted, it is certainly illuminating that the difference in the line counts between the java files and the cs files is so great, those numbers alone reveal the amount of infrastructure in the .NET platform that is in place to help developers concentrate on their problem domain instead of on making up for a lack of infrastructure.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Microsoft, Visual Basic, Visual Studio.Net, Visual C++ and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

We are required by the Sun Java Pet Store License to include the following legal information regarding the Java Pet Store application: